

强化学习中构建蜕变关系来检测学习策略的bug:

强化学习的基础:

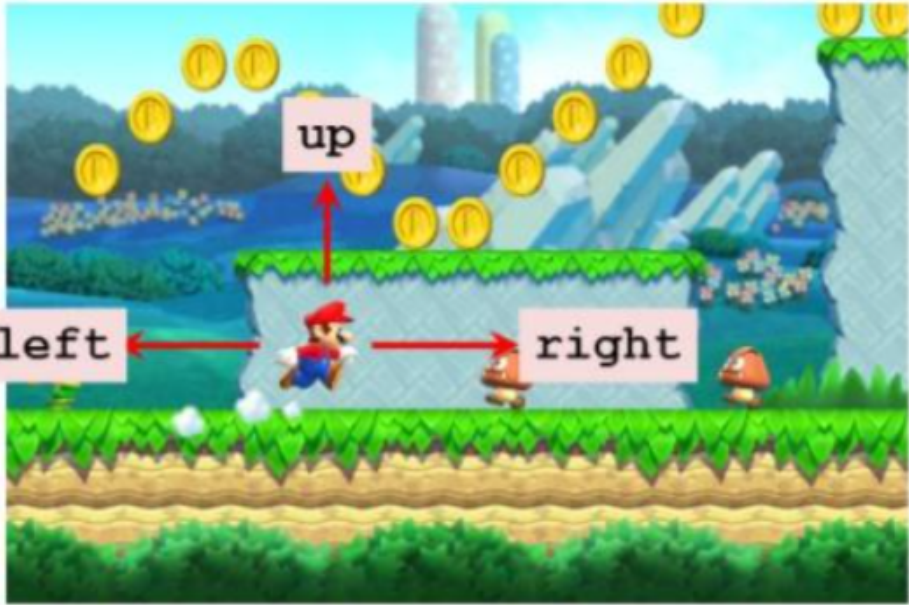
本篇论文涉及到了一些强化学习的内容，因此熟悉其基本概念相当重要。

强化学习的基本概念:

强化学习 (Reinforcement learning, RL) 讨论的问题是一个智能体(agent) 怎么在一个复杂不确定的 环境(environment) 里面去极大化它能获得的奖励。通过感知所处环境的 状态(state) 对 动作(action) 的 反应(reward), 来指导更好的动作, 从而获得最大的 收益(return), 这被称为在交互中学习, 这样的学习方法就被称作强化学习。

以马里奥的游戏的一张截图来解释一下这些概念

Terminology: policy



policy π

- Policy function $\pi: (s, a) \mapsto [0,1]$:
 $\pi(a | s) = \mathbb{P}(A = a | S = s)$.
- It is the probability of taking action $A = a$ given state s , e.g.,
 - $\pi(\text{left} | s) = 0.2$,
 - $\pi(\text{right} | s) = 0.1$,
 - $\pi(\text{up} | s) = 0.7$.
- Upon observing state $S = s$, the agent's action A can be random.

(出自知乎: [marsggbo](#))

- **state**: 这个就表示 t 时刻马里奥的状态, 用 s_t 表示。状态的定义方式也是可以由我们自己定义, 比如可以是马里奥所在的位置, 前后是否有怪兽等
- **action**: 知道了状态后, 我们需要做出决策或者说动作, 即**action**,用 a_t 表示。在这个游戏里, 动作有三个: up,left,right
- **policy**: 这个表示决策策略 $\pi(a_t | s_t)$, 它会决定在 s_t 时刻, 应该采取什么样的动作 a_t 。如果是均匀随机策略, 那么就表示每个动作被选择的概率相等, 即0.33。这当然不行, 所以我们需要设计更好的策略。
- **reward**: 强化学习里另外一个很重要的概念就是**奖励 (reward)**, 用 r_t 表示。对应到马里奥游戏就是迟到的金币越多奖励越多。这个reward需要预先设置或者用其他方法定义, 在这里你只需要知道在 s_t , 如果根据 π 采取 a_t 动作, 那么得到的奖励就是 r_t 。也就是说不同的动作会对应不同的奖励

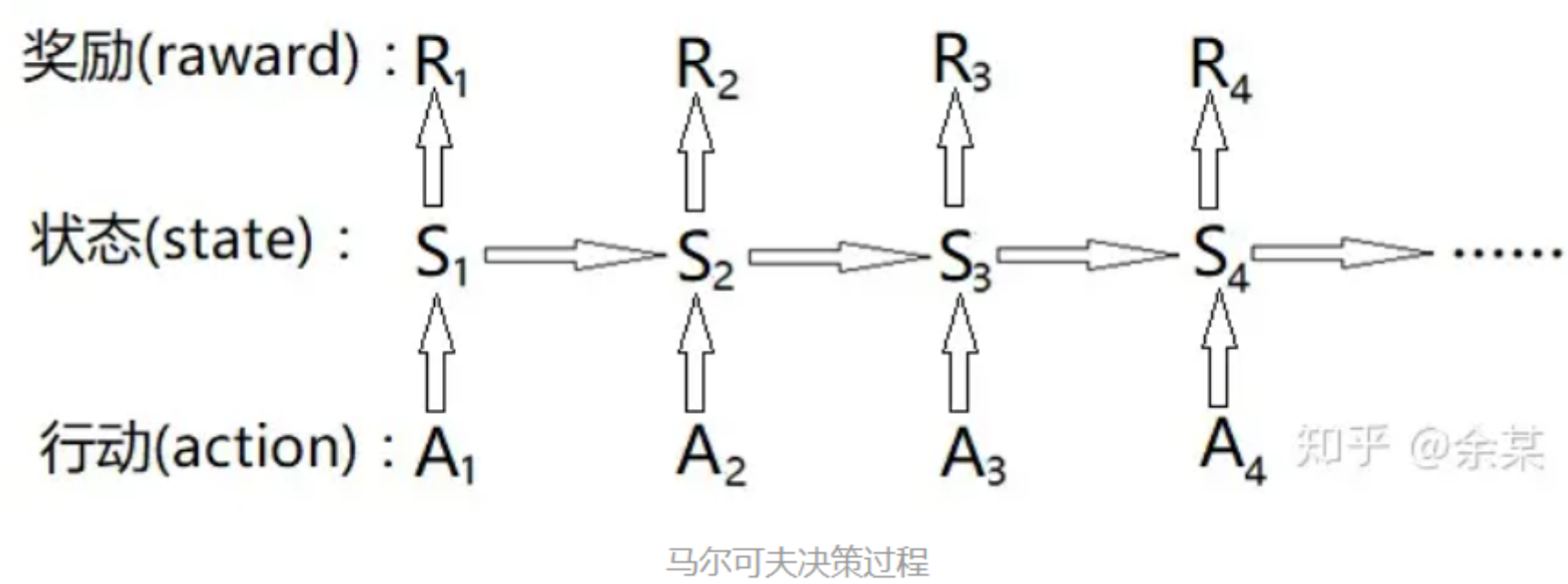
Reward和Return有点不同, 前者指的是每个时刻采取某种决策时候获得的即时奖励。而后者指的是自某个时刻采取的动作到游戏结束获得的总奖励。

马尔可夫决策过程：

[2.1.马尔可夫决策过程 \(MDP\) - 知乎 \(zhihu.com\)](#)

关于马氏链和马尔可夫决策过程

MDP:



摘要：

测试是一个很好的方法来验证某种学习策略的正确性（也可以是检测某个程序功能是否无误）。但是一般的测试方法很难去分析出错误的到底是由程序本身的bug还是由测试环境引起的。在软件测试时，如果正确的输出结果未知那么我们如何验证我们结果的准确性呢，这也是个大问题。而蜕变测试通过提出蜕变关系很好的解决了这个问题。而怎么确定蜕变关系在测定某个学习策略在特定环境下是否存在bug，这也是这篇论文要讨论的一个重点。

介绍：

如何确定一个蜕变关系：

本论文构建蜕变关系并不是直接通过预期输出行为来关系的构建的，而是构建的是状态 s 和 s' 之间的关系。

什么意思呢，首先我们的学习策略是确定一个状态如何行动，再根据行动确定下一个状态，一直这样下去直到获得最终的回报，那我们就可以通过改变状态，使其满足一定的关系，来观察能得到什么行动。

而这两种关系是通过松弛关系来形成联系的。比如说：相对于 s 状态， s' 状态下更容易完成任务。如果行动策略 π 在 s' 状态下行不通但是在 s 状态下能行得通，那就说明 π 在 s' 状态下存在bug。要构建这种松弛关系其实是很简单的。比如说如果前方存在障碍物，那么我们就可以通过移除障碍物来达到一个比较轻松的状态能通过障碍物（相当轻松）。而另一中状态下想要通过障碍物就显得没那么简单了。

？ ？ ？

Our key insight is to use the concept of relaxations to design novel oracles that do not require optimal policies.

测试状态的产生；

fuzzing模糊测试产生，通过随机产生对程序的输入，获取到不同的装态。

近似思想：

在近似思想中有两个近似分别为过近似（over-approximation）和下近似(Under-approximation).

记程序执行过程中所有可能出现的状态视为一个集合，称为实际状态集合（c），O-a一般来说是在c的一个超集O中分析，而U-a则是在c的一个子集中分析。一般来说下近似用来找错，上近似用来证明。并且上近似和下近似不能混用。

相关工作：

神经网络测试：

使用那些只关注神经网络的单一调用的测试技巧来测试强化学习效果不佳。因为测试方法都需要在有初始化状态的基础上，通过一系列相关的决定才能最终获得回报。而强化测试有许多个状态，对于每个状态都要分析一遍找到最优解这显然是不切实际的。并且有的测试方法只针对某个领域并需要较强的相关知识。而利用松弛来构建的松弛关系可以很好的解决这些问题。同时还制定了一个changing-nothing的规则，即如果对于一对满足松弛关系的s和s'，如果学习策略对s有效，那么对s'也有效，但是这个规则会对影响s'是否存在bug的判断。

安全强化学习：

即限定智能体的随机探索行为，以符合现实生活中的应用。

概念和符号界定：

- 1. $M = (S, A, T, S_0)$ 代表了一个马尔可夫决策的过程，其中s表示状态集合，A表示行为集合，T则是表示 $S \times A \rightarrow D(S)$ 的一个函数，即表示S状态发生A行为的概率*。而 S_0 则表示初始状态集合。
- 2.学习策略 Π :是一个输入一个状态得到一个行为的过程，追求的是一个状态或者状态转移时回报的最大化。当我们确定一个初始状态 s_0 时，那么这个策略的执行过程就是一个数列 $\sigma = s_0 \rightarrow a_0 \rightarrow s_1 \rightarrow a_1 \dots$ 一个状态可以根据策略在确定下一步的行为，由此再确定状态，形成循环直至完成任务。
- 3.学习策略的定义具有无记忆性和确定性。无记忆性体现在之前的状态和行为无法干预后面的行为和状态，后面一步的状态只与当前的行为和状态有关。而确定性则是说无论这个学习策略被执行多少次，只要给出了初始状态是一样的那么结果也是确定的。
- 4.假设 ϕ 是一个不应该出现的状态，如果再学习策略执行的过程 σ 中，如果出现了该状态那我们就可以认为这次执行是失败的，否则就是成功的。
 $P_\phi(\pi, s)$ 是在s状态下 Π 执行失败的概率， $P^*\phi(s)$ 记为所有学习策略失败的最小概率。
- 5.在本论文中使用状态变量的实数向量来表示当前的状态，每个状态s都能由一个确定的状态变量的向量 (v_1, \dots, v_n) 来唯一确定。
- 6.虚拟机，即环境environment,也叫E。
 $E.randomInit()$, 在初始状态池中随机返回一个状态作为初始状态。

$E.setState(s)$, 将 s 作为该环境的状态。 $s \in S$
 $E.step(s, a)$, 获取通过 $T(s,a)$ 函数获取最有可能的下一个状态。

Π -FUZZ框架概述：

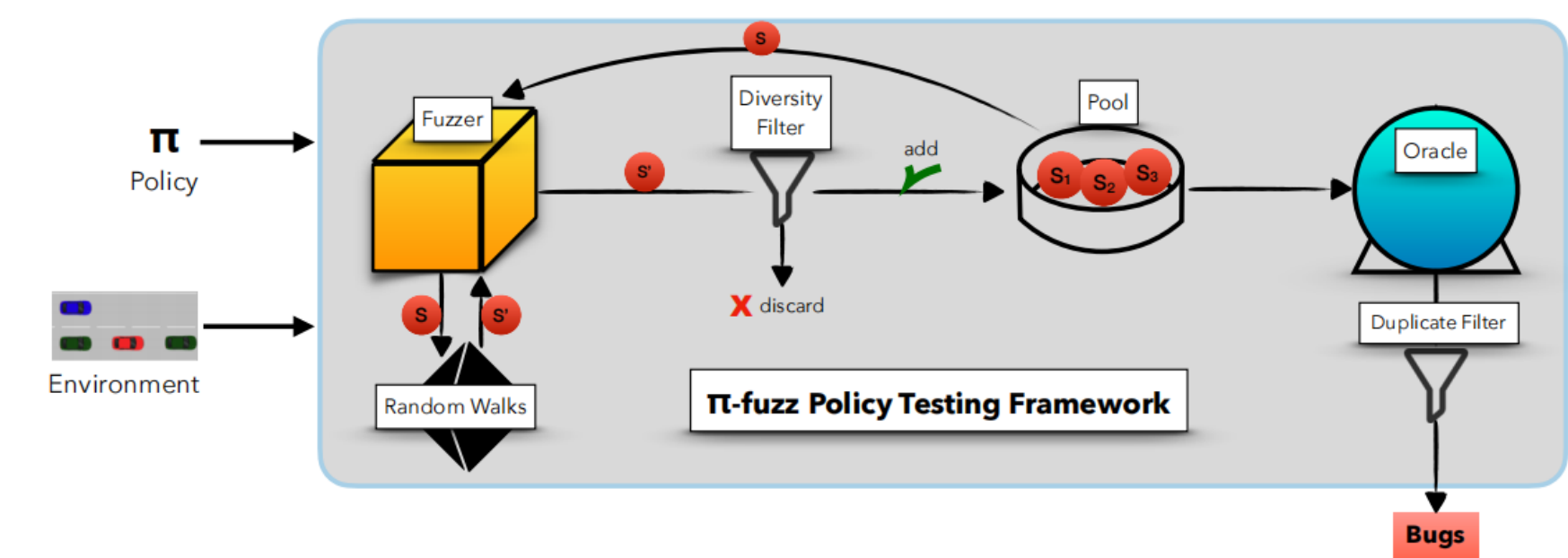


Figure 1: Overview of π -fuzz framework.

首先这个框架需要一个学习策略和环境作为输入，其次有两个重要的组成部分，Fuzzer和Orade。Fuzzer借助Random Walks随机游走算法根据一个状态随机生成另一个状态，再通过多样性检验（就是检测生成的状态是否早已存在）后加入到状态池中。而Orade则是分析在测试的状态 s 的情况下 Π 是否存在bugs。

学习策略的bugs：

本论文给出了两种类型的bug的定义：

Definition 1 (Bug). Let $M = (S, A, T)$ be an MDP, E a simulator for M , π a policy, and $s \in S$ a state.

- (i) We say that s is a **bug** in π if $P_\phi(\pi, s) > P_\phi^*(s)$.
- (ii) Given a random seed $\rho = r$, we say that the run $E^r.\sigma[\pi, s]$ is a **seed-bug** in π if $E^r.\sigma[\pi, s]$ fails, but there exists a policy π' such that $E^r.\sigma[\pi', s]$ succeeds.

bug1和bug2之间其实是有联系的，可以看成bug2其实是bug1的一个现实实例。

变质预言机设计：

松弛状态：

对于任意一个学习策略 Π ，若 s 的实现和 r 的实现在任意一个随机种子条件下都能成立，那我们就可以说 s 和 r 之间是松弛关系的状态对。
本质上来说就是 r 比 s 状态在同一个学习策略和随机种子下更容易成功。

Definition 2 (State Relaxation). Let $M = (S, A, T)$ be an MDP, and E a simulator for M . We say that $t \in S$ **relaxes** $s \in S$ if, for every policy π_s , there exists a policy π_t such that, for every random seed $\rho = r$, whenever $E^r.\sigma[\pi_s, s]$ succeeds, then $E^r.\sigma[\pi_t, t]$ also succeeds.

We say that $R \subseteq S \times S$ is a (state) **relaxation** if, for every $(s, t) \in R$, t relaxes s .

变质测试的预言机：

在一个MDP案例中，如果t,s是松弛关系，那么在执行学习策略的过程中如果出现：

- (i) $P\phi(\pi, s) < P\phi(\pi, t)$, 那么t就是学习关系 Π 中的一个bug。
- (ii) $E^r.\sigma[\pi, s]$ 成功执行 但是 $E^r.\sigma[\pi, t]$ 失败了, 那么 $E^r.\sigma[\pi, t]$ 就是一个seed-bug in π .

这就是本论文检测bug的理论基础。

论文中的变质预言机：

论文给出了三种不同的检验bug的function：

```
1 Function BUGORACLE( $R, \pi, s_i$ ):
2   evaluate  $P_\phi(\pi, s_i)$ ;
3   repeat ORACLE_BUDGET times
4      $t_i = \text{RANDOMSTATE}(\{t_i \mid R(t_i, s_i)\})$ ;
5     evaluate  $P_\phi(\pi, t_i)$ ;
6     if  $P_\phi(\pi, t_i) < P_\phi(\pi, s_i)$  then
7       return 1
8   return 0;

9 Function BASICSEEDBUGORACLE( $R, \pi, s_i, r$ ):
10  if  $E^r.\sigma[\pi, s_i]$  fails then
11    repeat ORACLE_BUDGET times
12       $t_i = \text{RANDOMSTATE}(\{t_i \mid R(t_i, s_i)\})$ ;
13      if  $E^r.\sigma[\pi, t_i]$  succeeds then
14        return 1
15  return 0;

16 Function EXTSEEDBUGORACLE( $R, \pi, s_i, r$ ):
17   $B = \{\}$ ;
18  if  $E^r.\sigma[\pi, s_i]$  succeeds then
19    repeat ORACLE_BUDGET times
20       $t_i = \text{RANDOMSTATE}(\{t_i \mid R(s_i, t_i)\})$ ;
21      if  $E^r.\sigma[\pi, t_i]$  fails then
22         $B = B \cup \{E^r.\sigma[\pi, t_i]\}$ ;
23  return  $B$ ;
```

这些算法中的R就是松弛关系

BugOracle算法检查由 π -fuzz生成的给定测试池状态 s_i 是否可以被识别为bug。它通过比较 $P\phi(\pi, s_i)$ 和 $P\phi(\pi, t_i)$ 的非松弛状态 t_i ，而是难度增加的关系即 $R(t_i, s_i)$ 。也就是说 t_i 比 s_i 更难实现。而评估 $P(s_i)$ 则只是一个子问题。

BasicSeedBugOracle 算法检验原理和Bugoracle基本相同，只不过检验的是seed bugs.

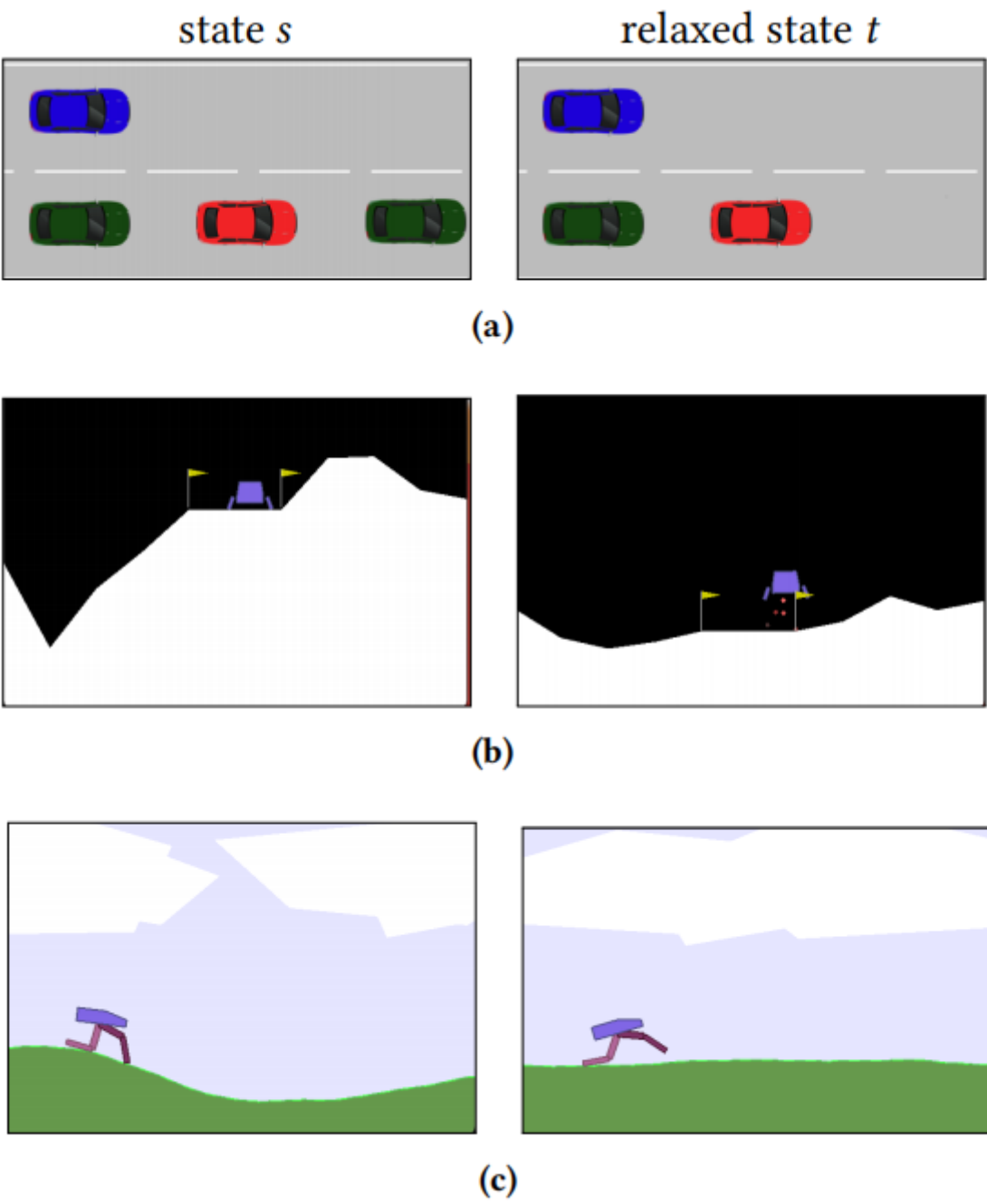
ExtBasicSeedBugOracle 算法是检验的拓展。如果在 $R(t_i, s_i)$ 下 s_i 表现良好，那我们可以通过 $R(s_i, t_i)$ 找出 t_i 中存在的seed bugs.

在此定义的基础上我们要对R进行抽样检测。为了保证能够找出明显的bugs (If π works well on s_i and t_i is much easier, it is unlikely that the policy is bad on t_i , thus not

leading to the detection of a bug. If π is bad on s_i and t_i is much harder than s_i , it is unlikely that the policy works well on t_i , again not leading to the detection of a bug.) 我们定义的R只包含一些简单的状态修改操作，即包括应用随机选择的变质操作（要么移走一个障碍物或者增加一个障碍物）与随机选择的参数。

如何获得松弛关系：

在预言机的设计过程中我们发现无论哪个检验函数都需要松弛关系。松弛关系的获取在某种情况下相当的简单。只需要简化一下完成任务的任务量就可以很好的实现这一点。



模糊算法的设计：

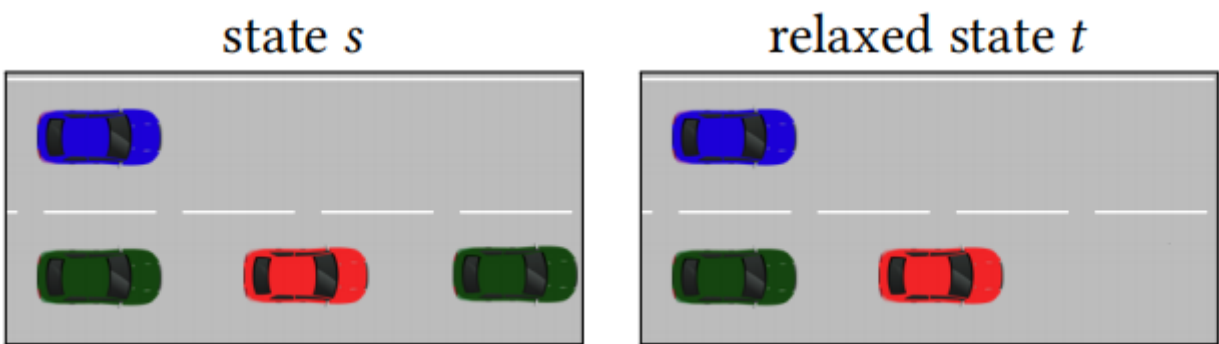
```
Algorithm 2: Fuzzing procedure
1 Function FUZZER(Env E, Policy  $\pi$ ):
2    $P = []$ ;
3    $P = \text{ADD}(E.\text{RANDOMINIT}(), P)$ ;
4   while  $\neg \text{INTERRUPTED}()$  do
5     if  $\text{RANDOMBOOLEAN}(\text{INC\_PROB})$  then
6        $s = \text{RANDOMSTATE}(P)$ ;
7     else
8        $s = E.\text{RANDOMINIT}()$ ;
9      $s' = \text{RANDOMWALK}(E, \pi, s)$ ;
10    if  $\min_{t \in P} d^{\text{Eucl}}(s', t) > \text{DIV\_THRESH}$  then
11       $P = \text{ADD}(s', P)$ ;
12    for each  $s_i \in P$  do
13      Run oracle on  $s_i$  (picking  $r$  if needed);
14 Function RANDOMWALK(Env E, Policy  $\pi$ , State  $s$ ):
15    $E.\text{SETSTATE}(s)$ ;
16    $k = \text{RANDOMINTRANGE}(0, \text{WALK\_LENGTH})$ ;
17   if  $\text{RANDOMBOOLEAN}(\text{POL\_PROB})$  then
18     repeat  $k$  times
19        $a = \text{RANDOMPOLICYACTION}(\hat{\pi}(s))$ ;
20        $s = E.\text{STEP}(s, a)$ ;
21   else
22     repeat  $k$  times
23        $a = \text{RANDOMACTION}(E.\text{actions})$ ;
24        $s = E.\text{STEP}(s, a)$ ;
25   return  $s$ ;
```

Fuzzer函数就是运行整个fuzzer架构的函数。首先先将E的初始状态加入状态池中，如何在不被外界打断的情况下（通过调用interrupted()函数打断），随后逐渐增加状态池中的状态量。这个增加的过程是随机的，通过INC_PROB这个参数的值来决定状态是取自状态池还是E的初始状态池中。随后便是调用randomWALK算法获取新生成的状态了。如果新生成的s'不在状态池P中，那么就加入状态池。 $d^{\text{Eucl}}(s, s')$ 这个函数计算的是s和s'之间的欧几里得距离（前面提到了我们的状态都是通过向量的形式表示的，也可以看成坐标）而DIV_THRESH为到池中已经存在的状态的最小距离设置了一个阈值。也就是说只有当欧几里得距离大于这个值的时候我们才说这个状态与池中的其他状态有一定的区别，因此可以加入状态池中。当状态池搭建完成后，就可以通过先前构建的预言机来测试这些状态了。

而RANDOMWALK算法则是生成状态的算法，首先根据POL_PROB这个参数的值选择状态行为的来源，一个是根据学习策略来选择最佳的行为，另一个则是从行为集合里面选择。都有步长的重复。

案例分析：

高速公路研究案例：

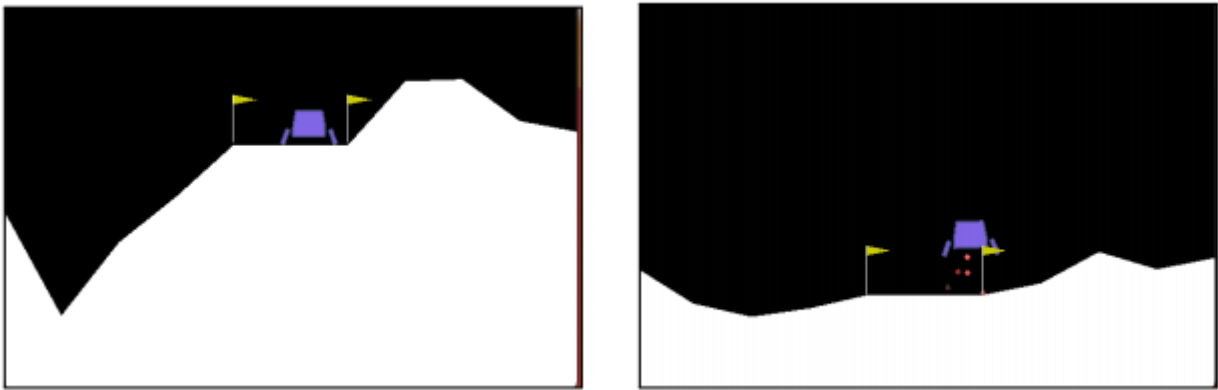


训练场景：高速公路上的两条道路，研究对象要从起点没有碰撞的到达终点，其他的车都有可以发生变道随后造成碰撞。研究对象有五种离散的行为，加速，减速，刹车，向左变道，向右变道。一旦发生碰撞，将以-100分的成绩结束。安全到达终点有100分的奖励，折算为0.95。

松弛关系：
减少研究对象的前车的数量，使其更容易到达终点。而通过增加前车的数量来加大难度。

参数设置： ???
WALK_LENGTH = 3
DIV_THRESH = 3.6

月球登陆：

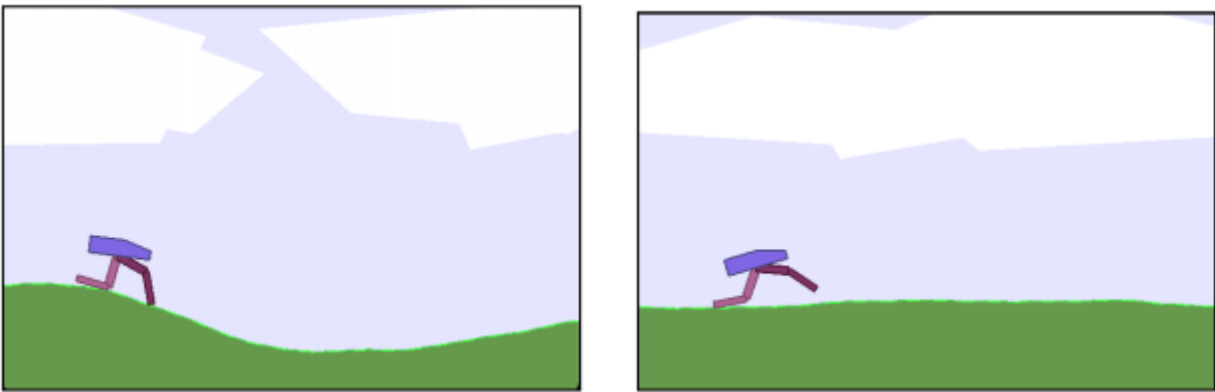


着陆器以一个随机的速度矢量出现在环境的顶部，而他的任务就是双腿着陆而其他部位不触碰到地面。如果其他的部位触碰到地面则判定为碰撞，任务失败。着陆器有四个离散的行为：启动底部发动机，启动左边发动机，启动右边的发动机，以及不启动发动机。将一条腿接触地面产生奖励+100，将身体接触地面产生奖励-100。没有折扣因素，最好的情况下的奖励是超过200

松弛关系：
松弛状态是通过降低表面的高度而产生的，给着陆器更多的时间来着陆。相反地，我们通过将表面高度增加到一个安全的距离来产生非松弛状

参数设置：
DIV_THRESH = 0.65
WALK_LENGTH = 25

双足行走：



一个双足机器人沿着一个表面粗糙的终点地形移动。机器人的任务是向前移动，直到地形结束。如果中途机器人摔倒认定失败。在这个环境下被观测者的行为集合是一个连续的四元组空间，每一个元组的取值范围都在0-1之间，代表的是每个关节的发力情况。每一种受力情况对应的行为是确定的。在这个实验中，除了地形的平缓程度和机器人的起始受力情况是随机的。

松弛状态：
由于光滑的表面更容易为步行者导航，松弛状态是通过使地形更平滑而生成的，而非松

弛状态是通过使地形更粗糙而生成的。

参数设置：

DIV_THRESH = 2.0

WALK_LENGTH =25

案例共性：

案例研究探索了简单动作动力学（高速公路）下的移动障碍，复杂动作动力学下的固定障碍（月球着陆器），以及与连续运动控制（BipedalWalker）保持平衡。因此，它们涵盖了广泛的应用，具有避障，这是一个在神经控制系统中普遍存在的问题。

实验

实验关心的是找到预言机判断bug的能力
同时还分析了DIV_THRESH，WALK_LENGTH 这两个参数的影响。

预言机：

在本实验中不仅仅只考虑变质预言机的效果，而是将八台预言机一起比较

MMBug, MMSeedBugBasic, MMSeedBugExt:

，这三台预言机代表的是算法1中的三个函数的实现。
这三台预言机是基于蜕变关系（在本论文中即松弛关系）的预言机

FailureSeedBug:

这个预言机就是把当 $Er.\sigma [\pi, si]$ 执行失败时，si判定为种子bug。

RuleSeedBug:

根据change-nothing规则，在一个松弛关系R(si,ti)中，如果 $Er.\sigma [\pi, si]$ 能否成功，那么对于ti也可以，就应该有 $\pi (si) = \pi (ti)$ 。

这个预言机就是通过这个规则来判断种子bug，如果在R(ti,si)中， $Er.\sigma [\pi, ti]$ 成功运行 and $\pi (si) \neq \pi (ti)$ ，那么我们就认为si时是个种子bug。

As $\pi (si) = \pi (ti)$ is possible but not necessary, this oracle may incorrectly classify si as a seed-bug? ? ? ? 这里有点疑惑，或许应该是不等号？
因此我们在这里只考虑真阳性的问题。

PerfectBug and PerfectSeedBug:

PeffectSeedBug用给定的随机种子探索来自给定状态的所有可能的轨迹，并只在存在获胜轨迹但策略从该状态失败时才检测到bug。因为我在确定下一个行为的时候其实是根据概率的大小来确定行为的，但是这个预言机就可以检测剩下的行为，看他们是否能有完成任务的路线，并检查其中的bug。

Perfectbug只是测量 $P\phi (\pi, s)$ 和 $P*\phi (s)$ ，如果 $P\phi (\pi, s) > P*\phi (s)$ ，则检测到一个bug。只适用于高速公路

MMSeedBug2Bug:

这个预言机会首先调用MMSeedBugBasic这个预言机，如果这时候MMSeedBugBasic把si标记为种子错误，那么接下来如果还有 $P\phi(\pi, ti) < P\phi(\pi, si)$., 那么MMSeedBug2Bug就会将si标记为一个bug。

不同预言机之间的关系：

RuleSeedBug \subseteq MMSeedBugBasic
MMSeedBugBasic \subseteq PerfectSeedBug, MMBug \subseteq PerfectBug
PerfectSeedBug \subseteq FailureSeedBug
MMSeedBug2Bug \subseteq MMSeedBugBasic, MMSeedBug2Bug \subseteq MMBug
MMSeedBug2Bug=MMSeedBugBasic=MMBug（这种关系适用于策略运行是唯一的确
定域）

实验结果：

预言机能力：

总的来说，研究表明，变质预言机对于识别行动策略中的bug非常有用。
另外外，种子bug检测是错误bug的一个实用代理，因为MMSeedBugBasic检测到的大多数种子bug都是bug(第一种)。特别是，MMSeedBugExt在识别不同状态的bug方面非常有效。

Fuzzer的参数配置：